1981

# Tracing COBOL programs- What can your students gain?

Henry Etlinger



Recommended Citation
NECC '84 6th Annual National Educational Computing Conference

# TRACING COBOL PROGRAMS— WHAT CAN YOUR STUDENTS GAIN?

**by Henry Etlinger**

*Rochester Institute of Technology*

In many introductory programming courses, students are urged to trace programs in a textbook to see if they understand how the programs work. In addition, students may have to trace simple programs on a test (a typical question calls for a display of a program's output). As a student confronts more and more complex language features, and as examples become more lengthy, the tracing idea seems to fall by the wayside. We expect a student to understand complex programs, but we don't seem to offer any practical guidance about how this understanding may be achieved.

**"Instead of using the tracing idea only for simple programs and for basic concepts, I would advocate using the tracing technique in complex situations as well."**

Instead of using the tracing idea only for simple programs and for basic concepts, I would advocate using the tracing technique in complex situations as well. A student needs a mechanism that offers a way of dealing with a novel program or new language features. This mechanism must be mechanical for a student to apply, yet offer the potential for gaining insight about a program's behavior. Tracing, in its many forms, could be such a mechanism, especially if a student has practiced it from the beginning of his or her studies.

We typically mean different things when we use the word "trace." We can think of tracing a program to a) find out what will be printed when a program is executed with a sample data file, b) show the values that some or all a program's variables take on during a program's execution, c) determine how many times particular instructions will be executed or how many times particular conditions will be examined, or d) show the basic flow of control through a program, noting how relevant indicators change and how files are processed. Through practice, most students can become adept at performing these and other types of traces. I will illustrate a tracing technique that has been used in an advanced COBOL class. This tracing technique comfortably fits with COBOL programs because COBOL programs have the characteristic of organizing the executable code into paragraphs (or sections). The technique, however, is also applicable to other programming languages, with modifications.

Before we look at a specific case, we must think of the motivations we can offer students so they will want to do traces. It's easy to think of negative features of tracing. First, students do not typically learn how to "read" programs. To understand a program, you need to see how it is "put together." You need to see the interactions between a program's components. By tracing a program, even a large program, with some sample data, a student will begin to see these interactions. What's more, tracing is an *active* process. All too often, I fear, students only look at a program superficially. Tracing makes students participate in the dynamic nature of a program.

Secondly, tracing is a fundamental tool for debug-

ging programs. When you stop to think of it, students spend most of their time working with incorrect or malfunctioning programs. Skillful tracing can help students rapidly locate that portion of a program where it begins to deviate from expectations. The faster they can find and correct their errors, the faster they can complete their assignments.

Besides the pragmatic issues, students who intend to become professional computer scientists ought to be exposed to some of the theoretical research on programming. As Ben Shneiderman has pointed out[1], program comprehension has different dimensions. Depending on the type of programming task, it may be more important to possess the skill of "forward" comprehension (determining the output for a given input) or "backward" comprehension (determining the input from a given output). Practicing both activities via traces may help students diagnose their own strengths and ultimately help steer them into appropriate career paths.

I often give the first tracing exercise in class with only a brief overview of what the exercise requires. Without fail, I find that many students lack the ability to systematically trace a program. To say it another way, they seem to have no method for performing the task—they don't know how to get started or they don't seem to be able to do the task carefully enough. This first exercise allows me to focus on what we are trying to accomplish in the trace and to suggest methods of recording information. After several more exercises, more students seem to at least understand the mechanics of tracing. Some students, however, still cannot do a trace and I would suggest that these students should seriously reconsider whether they want a career in programming.

Let's describe what a basic exercise might look like and then deal with variations. I would give students a program such as the one in appendix A[2]. I would give them some sample data for the relevant input fields. I would ask that they provide a trace of the paragraphs that would be entered as the program was executed with the sample data and I would request to see how relevant fields within the program changed. I might also ask a few questions about the program, questions that could be simply answered once the trace was completed.

The method of tracing advocated in class results in an "annotated, indented" list of paragraph names plus relevant information. Naturally, there are alternative schemes for displaying this information and students are encouraged to experiment with other notations. Many students seem happy to have one method presented to them and do not look for other ways to do things. I show below a sample trace for a small case.

| OLD-HOUSING-FILE | UPDATE-FILE |
|---|---|
| BILL A1020 | BILL A10252 |
| BOB A0715 | BOB A07153 |
| MARY A1025 | |

```
TRACE
400
    500
    720 [M—BILL A1020]
    740 [T—BILL A10252]
        (print report header)
520
    700
    740 [T—BOB A07153]
520
    (BILL to new master)
    720 [M—BOB A0715]
520
    (print successful drop)
    740 ['Y' to UP-END-SWITCH]
    720 [M—MARY A1025]
540
    600
        (MARY to new master)
    720 ['Y' to OLD-END-SWITCH]
HALT
```

The indentation allows a student to see how the components of a program fit together. Consistent indentation also provides a student with a concrete way of coming back from a series of procedure calls to the correct spot. I usually like to record relevant information about files right in the trace. With abbreviations like "M" for master file and "T" for transaction file, one can see at any point what the current records are that are being processed. Additional information, such as when particular records are written to the updated master file can also be added to the trace without detracting from its utility. Because all paragraph names are prefixed with a unique number, I allow students to simply record the leading number to save time.

Many common programming tasks (such as control break programs, sequential file processing programs, indexed file processing programs, and the like) lend themselves to this kind of analysis. I believe the indentation and the different patterns of numbers that emerge in the trace begin to show a student how a program handles different cases (such as changing records, going through a two-level control break, and the like). When a student first sees a program, he or she sees a static object and perhaps they do not clearly see how the pieces fit together.

As variations of this theme, I sometimes provide a list of the relevant fields and sometimes I expect students to determine this list. A COBOL program in particular contains many identifiers, not all which are relevant to tracing control flow. I urge students to examine all code carefully the first time they are led to that portion of a program. After seeing it one time, they ought to be able to trace through that portion more quickly the next time. In other words, they begin to see the "rhythm" of different parts of a program.

One major use I have found for this technique has been to compare programs that accomplish similar processing (such as sequential file updates). It's interesting for students to see that several programs may have different ways of accomplishing the same task. It's even more interesting for students to trace one such program (and have little difficulty doing so) and trace another program (and not be able to trace it completely accurately). This leads to an interesting discussion of the merits of the two programs and why one program is more difficult to understand than another.

In summary, tracing offers both instructors and students many benefits. Students can develop a method for learning how to read and more importantly, understand, programs. They can use this tool in debugging their own programs that may not work, and they can use this tool to test their understanding of new language features. Instructors can use tracing exercises as a way of comparing programs, pointing out style considerations, providing a framework in which test case design can be brought out, and testing a student's understanding of concepts, as well as specific language features.

## REFERENCES


1. Shneiderman, Ben, *Software Psychology: Human Factors in Computer and Information Systems*, Winthrop Publishers, Inc., 1980.
2. Beil, Donald H., *File Processing with COBOL: With Major Case Studies*, Reston Publishing Co., Inc., 1981.


### Appendix A

Below is a COBOL program used in the discussion of a tracing technique. The author gratefully acknowledges Reston Publishing Co., Inc., 11480 Sunset Hills Road, Reston, Virginia 22090, for permission to use this program from pages 174–176 of Donald H. Beil's text, *File Processing with COBOL*, 1981.

```
DATA DIVISION.

FILE SECTION.
FD  UPDATE-FILE
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS UPDATE-RECORD.
01  UPDATE-RECORD                  PIC X(80).
FD  OLD-HOUSING-FILE
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS OLD-FD-HOUSING-REC.
01  OLD-FD-HOUSING-REC             PIC X(80).
FD  NEW-HOUSING-FILE
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS NEW-HOUSING-REC.
01  NEW-HOUSING-REC                PIC X(80).

WORKING-STORAGE SECTION.
01  UPDATE-CHANGE-REC.
    05  UP-NAME                    PIC X(10).
    05  FILLER                     PIC X(5).
    05  UP-ROOM-NUMBER.
        10  UP-DORM                PIC X.
            88  VALID-DORM         VALUE 'A'.
        10  UP-FLOOR               PIC 99.
            88  VALID-FLOOR        VALUE 1 THRU 12.
        10  UP-DOOR                PIC 99.
            88  VALID-DOOR         VALUE 1 THRU 30.
    05  FILLER                     PIC X(4).
    05  UP-CODE                    PIC X.
        88  ADD-RECORD             VALUE '1'.
        88  CHANGE-RECORD          VALUE '2'.
        88  DROP-RECORD            VALUE '3'.
    05  FILLER                     PIC X(50).
01  OLD-WS-HOUSING.
    05  OLD-HOUSING-NAME           PIC X(10).
    05  FILLER                     PIC (5).
    05  OLD-HOUSING-ROOM-NUMBER.
        10  OLD-HOUSING-DORM       PIC X.
        10  OLD-HOUSING-FLOOR      PIC 99.
        10  OLD-HOUSING-DOOR       PIC 99.
    05  FILLER                     PIC X(60).
01  WS-SWITCHES.
    05  UP-END-SWITCH              PIC X.
        88  UP-END          VALUE 'Y'.
    05  OLD-END-SWITCH             PIC X.
        88  OLD-END         VALUE 'Y'.

PROCEDURE DIVISION.

400-DRIVER.
    PERFORM 500-INITIALIZE.
    PERFORM 520-UPDATE
        UNTIL UP-END OR OLD-END.
    PERFORM 540-TERMINATE-OUTPUT.
    STOP RUN.

500-INITIALIZE.
    OPEN INPUT OLD-HOUSING-FILE
        INPUT UPDATE-FILE
        OUTPUT NEW-HOUSING-FILE.
    PERFORM 720-READ-OLD-FILE.
    PERFORM 740-READ-UPDATE-FILE.
    DISPLAY '*** DROPS AND UPDATE ERRORS ***'
        UPON PRINTER.

520-UPDATE.
    IF UP-NAME > OLD-HOUSING-NAME
        WRITE NEW-HOUSING-REC FROM OLD-WS-
            HOUSING
        PERFORM 720-READ-OLD-FILE
    ELSE
        IF UP-NAME = OLD-HOUSING-NAME
            IF CHANGE-RECORD
                PERFORM 700-UPDATE-HOUSING-CODE
                PERFORM 740-READ-UPDATE-FILE
            ELSE
                IF DROP-RECORD
                    DISPLAY OLD-WS-HOUSING
                        '** SUCCESSFUL DROP **'
                        UPON PRINTER
                    PERFORM 740-READ-UPDATE-FILE
                    PERFORM 720-READ-OLD-FILE
                ELSE
                    DISPLAY UPDATE-RECORD
                        '** ADD ALREADY ON MASTER **'
                        UPON PRINTER
                    PERFORM 740-READ-UPDATE-FILE
```

```
        ELSE
            IF ADD-RECORD
                PERFORM 550-ADD-RECORD
                PERFORM 740-READ-UPDATE-FILE
            ELSE
                IF CHANGE-RECORD
                    DISPLAY UPDATE-RECORD
                        '** CHANGE NOT ON MASTER **'
                        UPON PRINTER
                    PERFORM 740-READ-UPDATE-FILE
                ELSE
                    DISPLAY UPDATE-RECORD
                        '** DROP NOT ON MASTER **'
                        UPON PRINTER
                    PERFORM 740-READ-UPDATE-FILE


540-TERMINATE-OUTPUT.
    IF UP-END
        PERFORM 600-WRITE-NEW-FROM-OLD UNTIL
            OLD-END
    ELSE
        PERFORM 620-WRITE-NEW-FROM-TRANS UNTIL
            UP-END.
    CLOSE OLD-HOUSING-FILE
        NEW-HOUSING-FILE
        UPDATE-FILE.

550-ADD-RECORD.
    MOVE UP-NAME TO OLD-HOUSING-NAME.
    MOVE UP-ROOM-NUMBER TO OLD-HOUSING-ROOM-
        NUMBER.
    WRITE NEW-HOUSING-REC FROM OLD-WS-HOUSING.
    MOVE OLD-FD-HOUSING-REC TO OLD-WS-HOUSING.

600 WRITE-NEW-FROM-OLD.
    WRITE NEW-HOUSING-REC FROM OLD-WS-HOUSING.
    PERFORM 720-READ-OLD-FILE.

620-WRITE-NEW-FROM-TRANS.
    IF ADD-RECORD
        PERFORM 550-ADD-RECORD
    ELSE
        DISPLAY UPDATE-RECORD '** NOT ON MASTER **'
            UPON PRINTER.
    PERFORM 740-READ-UPDATE-FILE.

700-UPDATE-HOUSING-CODE.
    MOVE UP-ROOM-NUMBER TO OLD-HOUSING-ROOM-
        NUMBER.

720-READ-OLD-FILE.
    READ OLD-HOUSING-FILE INTO OLD-WS-HOUSING
        AT END MOVE 'Y' TO OLD-END-SWITCH.

740-READ-UPDATE-FILE.
    READ UPDATE-FILE INTO UPDATE-CHANGE-REC
        AT END MOVE 'Y' TO UP-END-SWITCH.
```